Accessing Multiple Tables

JOIN

1. **Cross Join of Products and Suppliers:**

```
SELECT product_name, company_name
FROM products CROSS JOIN suppliers;
```

   - `CROSS JOIN`: Returns the Cartesian product of the two tables.
   - Fetches `product_name` and `company_name` by combining every product with every supplier.

2. **Inner Join of Products and Suppliers:**

```
SELECT product_name, company_name
FROM products
INNER JOIN suppliers
ON products.supplier_id = suppliers.supplier_id;
```

   - `INNER JOIN`: Returns only the rows that have matching values in both tables.
   - Fetches `product_name` and `company_name` where `supplier_id` matches in both `products` and `suppliers`.

3. **Retrieve Orders Ordered by Customer ID:**

```
SELECT order_id, customer_id
FROM orders
ORDER BY customer_id;
```

   - Fetches `order_id` and `customer_id` from `orders`, sorted by `customer_id`.

4. **Retrieve Customers Ordered by Customer ID:**

```
SELECT customer_id, company_name
FROM customers
ORDER BY customer_id;
```

- Fetches `customer_id` and `company_name` from `customers`, sorted by `customer_id`.

5. **Inner Join of Orders and Customers:**

```
SELECT orders.order_id, customers.company_name
FROM orders
INNER JOIN customers
ON orders.customer_id = customers.customer_id
ORDER BY orders.customer_id;
```

- Fetches `order_id` and `company_name` where `customer_id` matches in both `orders` and `customers`, sorted by `customer_id`.

6. **Distinct Employees Who are Sales Representatives:**

```
SELECT DISTINCT employees.first_name, employees.last_name
FROM employees
INNER JOIN orders
ON employees.employee_id = orders.employee_id
WHERE employees.title = 'Sales Representative' AND orders.ship_city IN
('Redmond', 'Seattle');
```

- `DISTINCT`: Removes duplicate rows from the result set.
- Fetches distinct `first_name` and `last_name` of employees who are sales representatives and have orders shipped to Redmond or Seattle.

7. **Distinct Customers from Specific Countries:**

```
SELECT DISTINCT customers.company_name,
  customers.contact_title,
  customers.city,
  customers.country
FROM customers
INNER JOIN orders
ON customers.customer_id = orders.customer_id
WHERE customers.country IN ('Mexico', 'Spain')
```

```
  AND customers.city <> 'Madrid'
  AND orders.shipped_date > '1997-01-01';
```

- Fetches distinct `company_name`, `contact_title`, `city`, and `country` of customers from Mexico or Spain, excluding Madrid, with orders shipped after January 1, 1997.

8. **Full Outer Join of Products and Suppliers:**

```
SELECT product_name, company_name
FROM products FULL OUTER JOIN suppliers
ON products.supplier_id = suppliers.supplier_id;
```

- `FULL OUTER JOIN`: Returns all records when there is a match in either left or right table.
- Fetches `product_name` and `company_name`, including unmatched rows from both `products` and `suppliers`.

9. **Left Join of Customers and Orders:**

```
SELECT customers.company_name, orders.order_id
FROM customers LEFT JOIN orders
ON customers.customer_id = orders.customer_id
ORDER BY customers.company_name;
```

- `LEFT JOIN`: Returns all records from the left table and the matched records from the right table.
- Fetches `company_name` and `order_id`, including customers with no orders, sorted by `company_name`.

10. **Count Orders by Employee for 1997:**

```
SELECT employees.first_name, employees.last_name, COUNT(orders.order_id)
FROM employees
INNER JOIN orders
ON employees.employee_id = orders.employee_id
WHERE orders.order_date BETWEEN '1997-01-01' AND '1997-12-31'
GROUP BY employees.last_name, employees.first_name;
```

- `COUNT`: An aggregate function that returns the number of rows that matches a specified condition.

- Fetches `first_name`, `last_name`, and the count of `order_id` for orders placed in 1997, grouped by employee names.

11. **Count Orders by Employee for Specific Dates in 1997:**

```
SELECT employees.first_name, employees.last_name, COUNT(orders.order_id)
FROM employees
INNER JOIN orders
ON employees.employee_id = orders.employee_id
WHERE orders.order_date > '1997-03-05'
  AND orders.order_date BETWEEN '1997-01-01' AND '1997-12-31'
GROUP BY employees.last_name, employees.first_name;
```

- Fetches `first_name`, `last_name`, and the count of `order_id` for orders placed after March 5, 1997, and within the year 1997, grouped by employee names.

12. **Left Join of Employees and Orders:**

```
SELECT employees.first_name, employees.last_name, orders.order_id
FROM employees
LEFT JOIN orders
ON employees.employee_id = orders.employee_id;
```

- Fetches `first_name`, `last_name`, and `order_id`, including employees with no orders.

13. **Right Join of Products and Suppliers:**

```
SELECT products.product_name, suppliers.company_name
FROM products
RIGHT JOIN suppliers
ON products.supplier_id = suppliers.supplier_id;
```

- `RIGHT JOIN`: Returns all records from the right table and the matched records from the left table.
- Fetches `product_name` and `company_name`, including suppliers with no products.

14. **Join Multiple Tables for Customers in France:**

```
SELECT c.company_name
FROM customers AS c
INNER JOIN orders AS o ON c.customer_id = o.customer_id
```

```
INNER JOIN order_details AS d ON o.order_id = d.order_id
INNER JOIN products AS p ON d.product_id = p.product_id
INNER JOIN suppliers AS s ON p.supplier_id = s.supplier_id
WHERE c.country = 'France' AND s.country <> 'France';
```

- Fetches `company_name` of customers in France who ordered products supplied by companies outside France.

15. **Left Join Suppliers, Products, and Categories:**

```
SELECT suppliers.company_name, products.product_name,
categories.category_name
FROM suppliers
LEFT JOIN products ON suppliers.supplier_id = products.supplier_id
LEFT JOIN categories ON products.category_id = categories.category_id;
```

- Fetches `company_name`, `product_name`, and `category_name`, including suppliers with no products and products with no categories.

Advanced SELECT Queries

1. **Find customers in the same city but different companies:**

```
SELECT a.company_name AS customer1, b.company_name AS customer2, a.city
FROM customers AS a, customers AS b
WHERE a.customer_id != b.customer_id AND a.city = b.city
ORDER BY a.city;
```

- `AS`: Renames a column or table with an alias.
- `!=`: Not equal operator.
- Fetches pairs of customer companies located in the same city but are different companies, sorted by city.

2. **Find employees and their managers:**

```
SELECT a.employee_id AS "Employee Id", a.first_name AS "Employee name",
  b.employee_id AS "Manager Id", b.first_name AS "Manager name"
FROM employees AS a, employees AS b
WHERE a.reports_to = b.employee_id;
```

- Fetches employee and their manager's IDs and names, where the employee reports to the manager.

3. **Find French customers in the same city but different companies:**

```
SELECT a.company_name AS customer1, b.company_name AS customer2
FROM customers AS a, customers AS b
WHERE a.country = 'France' AND a.company_name <> b.company_name AND a.city
= b.city;
```

- `<>`: Not equal operator.
- Fetches pairs of French customer companies located in the same city but are different companies.

4. **Find German suppliers in different cities:**

```
SELECT a.company_name AS customer1, b.company_name AS customer2
FROM suppliers AS a, suppliers AS b
WHERE a.country = 'Germany' AND a.country = b.country AND a.company_name
<> b.company_name AND a.city <> b.city;
```

- Fetches pairs of German supplier companies located in different cities and are different companies.

5. **Equijoin of Orders, Order Details, Products, and Customers:**

```
SELECT customers.company_name, products.product_name,
order_details.quantity
FROM orders, order_details, products, customers
WHERE orders.customer_id = customers.customer_id
AND products.product_id = order_details.product_id
AND order_details.order_id = orders.order_id;
```

- **`Equijoin`**: A join where the joining condition is based on equality between values in the common columns.
- Fetches company name, product name, and quantity for matched records across orders, order details, products, and customers.

6. **Count orders by French customers:**

```
SELECT c.company_name, COUNT(o.order_id)
FROM customers AS c, orders AS o
WHERE c.country = 'France' AND c.customer_id = o.customer_id
GROUP BY c.company_name;
```

- `GROUP BY`: Groups rows that have the same values into summary rows.
- Fetches company name and count of orders for French customers, grouped by company name.

7. **Count French customers with more than one order:**

```
SELECT c.company_name
FROM customers AS c, orders AS o
WHERE c.country = 'France' AND c.customer_id = o.customer_id
GROUP BY c.company_name
HAVING COUNT(o.order_id) > 1;
```

- `HAVING`: Used to filter records that work on aggregated data.
- Fetches company names of French customers with more than one order.

8. **Non-equijoin of Orders and Employees:**

```
SELECT order_date, employees.first_name || ' ' || employees.last_name AS
full_name,
  employees.hire_date
FROM orders, employees
WHERE orders.order_date > employees.hire_date;
```

- `Non-equijoin`: A join where the joining condition is based on a non-equality condition.
- Fetches order date, full name, and hire date where order date is after hire date.

9. **Detailed Order Information:**

```
SELECT c.company_name AS customer,
  p.product_name AS product,
  od.quantity,
  od.unit_price AS price,
  ROUND((od.quantity * od.unit_price)::numeric, 2) AS total
FROM customers AS c
JOIN orders AS o ON c.customer_id = o.customer_id
```

```
JOIN order_details AS od ON o.order_id = od.order_id
JOIN products AS p ON od.product_id = p.product_id;
```

- `JOIN`: Combines rows from two or more tables based on a related column.
- `ROUND`: Rounds a number to a specified number of decimal places.
- Fetches customer, product, quantity, unit price, and total price for each order.

10. **Total Sales by Customer:**

```
SELECT c.company_name AS customer,
  SUM(od.quantity * od.unit_price)::numeric(30, 2) AS total
FROM customers AS c
JOIN orders AS o ON c.customer_id = o.customer_id
JOIN order_details AS od ON o.order_id = od.order_id
JOIN products AS p ON od.product_id = p.product_id
GROUP BY c.company_name
ORDER BY total DESC;
```

- `SUM`: An aggregate function that returns the sum of a numeric column.
- Fetches total sales for each customer, grouped by company name, and ordered by total sales in descending order.

11. **Customers Who Ordered a Specific Product:**

```
SELECT c.company_name
FROM customers AS c
JOIN orders AS o ON c.customer_id = o.customer_id
JOIN order_details AS d ON o.order_id = d.order_id
JOIN products AS p ON d.product_id = p.product_id
WHERE p.product_name = 'Tofu';
```

- Fetches company names of customers who ordered 'Tofu'.

12. **Distinct French Customers with Foreign Suppliers:**

```
SELECT DISTINCT c.company_name
FROM customers AS c
JOIN orders AS o ON c.customer_id = o.customer_id
JOIN order_details AS d ON o.order_id = d.order_id
JOIN products AS p ON d.product_id = p.product_id
```

```
JOIN suppliers AS s ON p.supplier_id = s.supplier_id
WHERE c.country = 'France' AND s.country <> 'France';
```

- `DISTINCT`: Removes duplicate rows from the result set.
- Fetches distinct company names of French customers who ordered from foreign suppliers.

13. **Distinct French Customers with French Suppliers:**

```
SELECT DISTINCT c.company_name
FROM customers AS c
JOIN orders AS o ON c.customer_id = o.customer_id
JOIN order_details AS d ON o.order_id = d.order_id
JOIN products AS p ON d.product_id = p.product_id
JOIN suppliers AS s ON p.supplier_id = s.supplier_id
WHERE c.country = 'France' AND s.country = 'France';
```

- Fetches distinct company names of French customers who ordered from French suppliers.

14. **Customers and Suppliers in the Same City:**

```
SELECT customers.company_name AS customer,
  suppliers.company_name AS supplier,
  customers.city
FROM customers, suppliers
WHERE customers.city = suppliers.city;
```

- Fetches company names of customers and suppliers located in the same city.

15. **Union of Two Constant Values:**

```
SELECT 23 AS test
UNION
SELECT 45 AS test;
```

- `UNION`: Combines the result sets of two or more `SELECT` statements, removing duplicates.
- Combines two constant values (23 and 45) into one result set.

16. **Union of Cities from Customers and Suppliers:**

```sql
SELECT city FROM customers
UNION
SELECT city FROM suppliers
ORDER BY city DESC;
```

- Combines and fetches distinct cities from customers and suppliers, sorted in descending order.